4675P007 PATENT

UNITED STATES PATENT APPLICATION

for

# METHOD FOR PERFORMING RESPONSE TIME ANALYSIS OF NETWORK PERFORMANCE

Applicant(s):

Steven J. Schaffer
Jacob Weil

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Boulevard
Los Angeles, CA 90026-1026
(303) 740-1980



## EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: EL750127900US
Date of Deposit March 5, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Debbie Peloquin
(Typed or printed name of person mailing paper or fee)

Debbie Peloquin
(Signature of person mailing paper or fee)

# METHOD FOR PERFORMING RESPONSE TIME ANALYSIS OF NETWORK PERFORMANCE

## COPYRIGHT NOTICE



## FIELD OF THE INVENTION

**[0002]** This invention relates to the field of network computing, and more particularly, to a method and system for monitoring network, client, and server performance.

## BACKGROUND OF THE INVENTION

**[0012]** One way to monitor network performance is by measuring the processing time on a first node, such as a client, and the processing time on a second node, such as a server. In earlier versions, the client and server were on the same LAN (local area network), so that factors such as network delay did not need to be considered.

**[0013]** To accommodate more realistic implementations, network delay should also be considered when monitoring network performance.

## SUMMARY OF THE INVENTION

In one aspect of the invention, a method for monitoring network performance is disclosed. The method monitors a flow having one or more frames within a thread by calculating a node's active time, including the amount of time each of the frames is processed on a sending node in a network and the amount of time each of the frames is processed on a receiving node in the network; and the amount of time each of the frames is in transit across the network.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0003]** The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

**[0004]** FIG. 1 illustrates an Application Performance Report in a Chart format.

**[0005]** FIG. 2 illustrates an Application Performance Report in a Details format.

**[0006]** FIG. 3 illustrates an example of Request Preparation and Reply Preparation processing types.

**[0007]** FIG. 4 illustrates one exemplary embodiment.

**[0008]** FIG. 5 illustrates an example of a flow.

**[0009]** FIG. 6 illustrates an example of a multi-tier algorithm.

## DETAILED DESCRIPTION OF THE INVENTION

**[0010]** The present invention includes various operations, which will be described below. The operations of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the operations. Alternatively, the operations may be performed by a combination of hardware and software.

**[0011]** The present invention may be provided as a computer program product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs (Read Only Memories), RAMs (Random Access Memories), EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electromagnetic Erasable Programmable Read Only Memories), magnetic or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions. Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way

of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection). Accordingly, herein, a carrier wave shall be regarded as comprising a machine-readable medium.

Introduction

**[0008]** Response Time Analysis (hereinafter RTA) produces underlying data that is presented to a user in several reports, including the Application Performance Report (Chart and Details), Node Processing Time Report, and Flows Report. The RTA algorithms and techniques are the underlying technology that produces information that is presented in the GUI (Graphical User Interface).

**[0009]** The user's first exposure to the response time analysis is in the high-level summaries presented in the health report. Those summaries give the user quick information without pouring through the details. Many of the details are then made available in the Node Processing Time report and Flows report.

**[0010]** This document contains a thorough description of the algorithms and techniques used in the underlying response time analyzer.

Application Performance Report

*Chart*

[0012] FIG. 1 illustrates an Application Performance Report. The four summary results shown in the first panel of the application performance report are:

Network Busy Time

[0013] The network busy time 100 is the total time that one or more meaningful frames are in transit across the network. Network Busy Time is computed and reported for both network directions (from primary to secondary and vice-versa). After the Flow concept has been introduced the Network Busy Time will be revisited to describe which frames are meaningful. Not all frames traverse the network. Only frames that are exchanged between the two capture points are included in the Network Busy Time. Consequently, Network Busy Time is applicable only with a multi-segment merged or adjusted trace. A trace is a sequence of frames that have flowed between two or more nodes over the time period of a capture (to be discussed below). Traces can be merged and adjusted, for example. The Network Busy Time can be broken down into two parts:

[0014] **Insertion Time (sec):** The cumulative time it took for the frames to be inserted into the network. In a merged or adjusted trace, the insertion time for each frame is computed as AdjustedBytes * 8 / Bandwidth. The network bandwidth utilization is computed for each direction of the network. Only frames that are known to have traversed the network are included. The term AdjustedBytes is used in the above expression to indicate the bytes that would

have occurred at the point the frame crossed the WAN link. The capture environment allows the user to specify if the size of frames should be adjusted to compensate for the different WAN headers. If the user chooses to adjust the frames, then AdjustedBytes = Bytes(as captured) – DLCHeader(as captured) + DLCHeader(specified in capture environment)

**[0015]** **Network Queueing, Propagation & Processing (QPP) Time (sec)**: The portion of the time that frames were in the network that is caused by queueing (in routers), processing and propagation. This is the portion of the total transit time that is not counted as insertion time. Throughout the description, this term is referred to as QPP time.

Node Active Time

**[0012]** There is one bar for each node in the Node Active Time 102, showing the portion of the task duration for which the node was active, either processing or sending. Each bar is broken down into two components:

**[0016]** **Node Processing Time**: For each node, the overall node processing time is a measure of the amount of time that the node was processing during task. A task is a user-invoked operation that creates network traffic in the course of its execution and ends with a screen update or other acknowledgement on the user's machine. A task, once invoked, executes without further user intervention until it has finished processing.

**[0017]** For single threaded applications, this is simply the sum of the individual node processing times (described later). If an application can have

multiple requests outstanding to a server (such as the typical Web browser), then the node is considered processing if it is processing one or more requests. The Node Processing Time for any node cannot be greater than the duration of the task, whereas the sum of the Processing Times for all nodes can be larger than the task's duration if there are parallel (overlapping) threads.

**[0018]** **Node Sending Time**: For each node, the overall node sending time is the amount of time that the node is in the process of sending data, but is not otherwise processing. If a node is in the process of sending a set of frames, then the node is considered to be in the sending state, but only if it is not processing another request at the same time. Sending time is important because it could indicate that the node is processing in order to prepare the remaining frames, or it could be caused by other factors that do not indicate that the node was processing. The most likely other factor is that the network is heavily utilized and the node cannot send all of its data at once. Other causes include an insufficient TCP window size, the normal slow-start nature of TCP, inability of the receiving node to remove the data from the TCP buffer fast enough, or an inefficient TCP implementation at either the sending or receiving node.

**[0019]** From the Application Performance Chart, double-clicks will result in the following drill-downs:

**[0014]** On a Node Processing Time portion of a node bar: Brings up the Node Processing Detail report filtered/highlighted on the specified node.

**[0015]** On a Node Sending Time portion of a node bar: Brings up the Node Sending Detail report filtered/highlighted on messages sent by the specified node.

**[0016]** On either network bar: Brings up the Network Utilization and Latency graph.

*Details*

**[0020]** An example of an Application Performance Report – Details is shown in FIG. 2. The major sections of the detailed report are described in the following sections.

Overall Summary

**[0021]** The purpose of the overall summary 200 is to give the user info on the task duration, any errors that were detected, the capture environment, and a very terse summary of the conversations, threads and turns. The information in the overall summary can alert the user to errors or the fact that they may not have captured what they intended to capture (for example, if the task time, number of conversations, or number of threads isn't what the user expects).

**[0022]** The values displayed in this section are:

**[0017]** **Task Time** (sec): The duration of the task. This should be equivalent to the "stopwatch time" of the task. If it is not, then portions of the time may be missing or may need to be deleted.

**[0018]** **Traffic Duration** (sec): The span of time during which frames exist. The user can click on the label to open the bounce diagram, which will show them all of the frames and the time period that they span.

**[0019]** **Errors**: A graphical summary of the number of errors, warnings, and informational errors detected in the trace. A click on the Errors label opens the Error Report.

**[0020]** **Capture Environment**: This is a legend, so to speak, for the meaning of the arrows that are in many other places in the report. A capture is a process where a traffic collector or capture agent listens for and collects network frames exchanged between nodes in a distributed application and stores that data for off-line analysis. The legend keys are taken from the Capture Environment specified by a user. The primary capture location is on the left and the secondary location is on the right.

**[0021]** **Conv**: Conversations, both the total for the task, and those that are between one node on the primary and another in the secondary location, indicated by the "<-->" label. Hereafter, the phrase "conversations that traverse the network" means conversations for which one node is in the primary location and the other is in the secondary location.

**[0022]** **Threads**: The number of threads in the task (Total) and in the conversations that traverse the network between the primary and secondary capture points. A thread is a sequence of frames that constitutes a single application or

protocol action. For example, the retrieval of a graphic from a WWW (World Wide Web) server is a thread. A thread will always be between a pair of nodes.

**[0023]** **Turns**: The number of turns in the task, and in the conversations that traverse the network between the primary and secondary capture points. <--> Turns specifies the sum of the turns for threads that have one node in the primary location and the other node in the secondary location.

**[0024]** **Bytes/Turn**: The average number of bytes in each turn, both as a total for the task and for the conversations that traverse the network.

**[0025]** **Frames/Turn**: While not shown, this characteristic can also be specified. It defines the average number of frames in each turn, both as a total for the task and for the conversations that traverse the network.

Traffic

**[0023]** The traffic section 202 provides the user with an overall summary of several traffic measures, for the entire task (the Total row), over the network (the <--> row), and in each direction across the network. As used herein, "over the network" will mean in both directions between the primary and secondary capture points. Columns in this section include:

**[0026]** **Bytes**: Sum of bytes for all frames in each classification. For the network classifications, the byte counts for each frame will be adjusted for the DLC header size if the user has chosen to perform this adjustment in the capture environment.

**[0024]** For <-->, → and ← Bytes, the value is the number of bytes that crossed the point of contention in the network as specified in the Capture Environment for the task. If the user did not choose to adjust frames for DLC header in the capture environment, then Network Bytes = Bytes for each frame. If the user did choose to adjust frames for DLC header in the capture environment, then Network Bytes = Bytes – DLC Header(as captured) + DLCHeaderbytes(specified in the capture environment)

**[0027]** **% of <--> Bytes**: Applicable only to the two directions of the network, this column shows what percentage of the bytes that traversed the network are attributed to each direction. The two values will add to 100% (subject, of course, to rounding in the display).

**[0028]** **Frames**: The total number of frames in the task (top row), and the number of frames that should have crossed the network, whether they were contained in both captures or not. If they weren't, such will be reported in the two Frames Missing columns at the right of this section. A frame is a collection of bits comprising data and control information (overhead) that is transmitted between nodes in a network.

**[0029]** **Avg Frame**: The average frame size, in bytes. Avg Frame = Bytes / Frames.

**[0030]** **Captured Load** (kbps and %): The captured load is the average rate, in kbps and as a % of the total bandwidth, of the frames that traversed the network in each direction. The adjusted bytes (refer to the discussion above) are used.

**[0031]** **Frames Missing at Source**: The number of frames that should have been in the trace from the capture point where the sending (source) node was placed. Frames missing at the source are usually an indication that they should have been captured but were not. This can be caused by the capture beginning too late or ending too early, or by the inability of the capture device to capture all of the frames.

**[0032]** **Frames Missing at Destination**: The number of frames that should have been seen in the trace from the capture point where the receiving (destination) node was placed. Frames missing at the destination could be lost in the network due to congestion or a failure of a network component, or they could be missing for the same reasons that frames missing at the source can be missing.

**[0025]** There can be situations where not all of the frames that should have traversed the network were actually captured, thereby resulting in missing frames. For instance, one of the captures may have started before or ended after the other and may contain frames that traversed the network. However, since those frames are not in the other capture, it is not known if they actually traversed the network. Such frames will be flagged with an error of Lost Frame or Dropped Frame, respectively, depending on whether it was captured only at its source segment or destination segment.

**[0026]** If it is assumed that missing frames really did traverse the network, then their bytes/frames/threads/conversations are included in the <--> metrics. There is no way to know whether a frame that is missing at the destination actually consumed bandwidth at the network contention point before being

dropped. Thus, the worst case is assumed – that it did consume network bandwidth, and missing frames are also included in the <--> metrics.

Network Busy Time

**[0027]** The network busy time section 204 helps the user determine how busy the network was during the task and how that busy time breaks down into insertion time and QPP time.

**[0028]** The network busy time section 204 is presented for merged tasks and single-trace adjusted tasks. There are two rows for the network metrics, one for the primary to the secondary location (→), and the second from the secondary to the primary location (←).

**[0029]** The columns in this section correspond exactly to the portions of the network bars in the Application Performance Report graph.

**[0033]** **Insertion Time** (sec and %): The cumulative time that it took to insert the captured frames into the network at the point of lowest bandwidth specified by the user. The insertion time is based on the network bytes of each frame should the user decide to adjust for DLC headers. of the network by the captured bytes. The bandwidth util in kbps is: Bytes that traversed the network in the specified direction * 8 / (duration of the task in seconds * 1000). The bandwidth util in % is (the bandwidth util in kbps / capacity of the link in kbps) expressed as a percentage. The capacity of the link is specified by the user in the capture environment.

**[0034]** **QPP Time** (sec and %): Queueing, Processing and Propagation time.

**[0035]** **Total** (sec and %): The total time that one or more meaningful frames was traversing the network. Meaning frames include all data frames and TCP acknowledgements that are within the data portion of a message. Meaningful frames do not include TCP acknowledgements that are sent after the last data frame in a message is sent.

Network Frame Transit Statistics

**[0030]** The network frame transit statistics section 206 comprises 2 rows, one for each direction of the network as described above.

**[0031]** This section includes only meaningful frames. As such, the statistics do not include TCP acknowledgements that occur after a message.

**[0036]** **Transit Time** (sec): the first column will contain the graphical depiction of the min, avg, and max frame transit time. The transit time of each frame is the difference between its Time Sent and Time Received.

**[0037]** **Transit Time Min** (sec): The transit time of the frame that has the smallest transit time.

**[0038]** **Transit Time Avg** (sec): The average (mean) transit time for the meaningful frames.

**[0039]** **Transit Time Max** (sec): The transit time of the frame that has the largest transit time.

**[0040]** **Transit Time Frame Count**: The number of frames included in the transit time statistics. Equivalent to the number of meaningful frames that were

captured at both sides (or adjusted). Note that this number is equal to or less than the number of frames that traversed the network in the specified direction. It does not include TCP acknowledgements after a message or frames that are missing at the source or destination.

**[0041]** **Latency Statistics** (min, avg, max): While not shown, statistics on the latency of meaningful frames that traverse the network may be shown. As described before, meaningful frames are data frames and the TCP acknowledgements that occur during the data transfer portion of a flow.

**[0042]** **Overlap Avg** (Frames): The average number of frames that are in transit when there is at least one frame in transit. It is a measure of the application's ability to send more than one frame at a time and the network conditions requiring the application to do so.. Put another way, it is the average number of frames sent by the application when the application has sent frames. Higher values mean the application is less susceptible to network latency and bandwidth.

**[0043]** **Overlap Max** (Frames): The largest number of frames that are in transit in the network at any given time.

Node Active Time

**[0032]** This is a tabular format of the node bars (processing and sending times) that were described earlier in the Application Performance Report Chart.

Node Processing Statistics

**[0033]** Statistics on the node processing periods.

**[0044]** **Processing Time** (sec): A graphical depiction of the min, avg and max node processing period.

**[0045]** **Processing Min** (sec): The shortest node processing period.

**[0046]** **Processing Avg** (sec): The average node processing period.

**[0047]** **Processing Max** (sec): The longest node processing period.

**[0048]** **Processing Periods**: The number of processing periods is important as it tells the user if there are a large or small number of node processing periods.

**[0049]** **Overlap Avg**: The average number of processing periods that occur simultaneously at the node during the times that the node is in at least one processing period. A value of 1.0 means that the node never processed more than one request at a time. This value cannot be smaller than 1.0.

**[0050]** **Overlap Max**: The largest number of processing periods occurring at any given instant.

Node Processing Detail Report

**[0034]** In addition to the overall summary results presented in the Application Performance Report, the RTA also identifies and reports several sets of details. One of these is the node processing detail.

**[0035]** Each node processing time is one component in the Overall Node Processing Time. The UI allows one to see the Individual Node Processing Times for all nodes or for one node at a time. Note that since individual node

processing times can overlap, the sum of the individual node processing times can be greater than the Overall Node Processing Time for a node.

**[0036]** The attributes of each node processing time (aka columns in the Node Processing Detail Report) are:

**[0051]** **Node Name**.

**[0052]** **Node Address**.

**[0053]** **Errors**. The number of errors associated with either the start or end frame. The user can drill into to the error report to see the errors.

**[0054]** **Duration** (sorted descending by default). The time span of the processing time, in seconds.

**[0055]** **Processing Type**. One of the list specified below

**[0056]** **Start Time**. The time at which the node began processing

**[0057]** **Start Frame**. The frame number at the beginning of the processing time. The user can drill down to the bounce or packet trace which will take them to the start frame.

**[0058]** **End Time**. The time at which the node stopped processing.

**[0059]** **End Frame**. The frame number that is at the end of the processing time. The user can drill down to the bounce or packet trace which will take them to the end frame.

[0060] **Start Frame Description**. The description (decode) of the start frame.

[0061] **End Frame Description**. The description (decode) of the end frame.

[0037] For most of the node processing types, there are corresponding processing types for client and server. For example, Client Processing and Server Processing. They basically have the same meaning, but the "Client" one is assigned when the node is the client in a thread whereas the "Server" one is assigned when the node is the server in the thread. Each node processing type has an internal code number that is listed in parentheses. This code number never appears in the GUI. The node processing types and their meanings are:

[0062] **Client Before Thread** 300: The time period prior to the first data frame in the thread when that first data frame is sent by the client. The processing time extends back to the previous data frame that was received by the node or the beginning of the task if there isn't one.

[0063] **Client Processing**: Within a thread, when the client node sends out a subsequent request, the time before that request is considered to be a client processing time. The time extends back to the previous data frame that was received by the client. Note that that data frame can be on the same thread or it can be on another thread.

[0064] **After last frame** 308: The time period from the last data frame to the end of the task is always assigned to the client node for the task. You can reassign the client node in the conversation map. This processing type will always end at the time that is the end of the task.

**[0065]** **Server Before Thread**: This is defined in the same way as Client Before Thread, but occurs if the first data frame in the thread is sent by the server of the thread. Normally one would not expect the server of the thread to send the first data frame (since the client normally initiates activity by sending a request) but if the capture starts in the middle of a thread, or if the client and server are assigned incorrectly then you may get this processing type. Note that the Thread Analysis window comprises a command to swap the client and server of a thread if they are identified incorrectly.

**[0066]** **Server Processing** 304: Within a thread, this is the time period from the point that the last frame in a request is received by the server to the time that the first response frame is returned by the server. During this time the server is assumed to be processing the request and consequently this time is identified as a server processing time. Note that with multi-tier applications there are cases where an upper-tier request may interrupt a server processing time. Multi-tier is described later.

**[0038]** **Request Preparation** 302: In multi-tier, when a mid-level server processes after receiving a request from a lower tier until it begins sending a subsequent request to another server. FIG. 3 shows this processing type at the App Server between frames 1 and 3.

**[0067]** **Reply Preparation** 306: In multi-tier, when a mid-level server processes after receiving a reply from another server until it begins sending its reply to its requesting node. FIG. 3 shows this processing type at the App Server between frames 5 and 7.

Flows Report

**[0039]** A flow is a set of data frames that is sent from one node to another. Specifically, a flow comprises only frames in a single thread, and spans a time period where there are no data frames traveling in the opposite direction. A flow also includes the TCP acknowledgements that are sent in the opposite direction during the flow and before the direction of data transmission reverses.

**[0040]** Meaningful frames were discussed earlier. All data frames, and TCP acknowledgements within a data flow are meaningful frames. TCP acknowledgements that occur after the last data frame in a flow are not meaningful frames for the purpose of network busy time computation.

**[0041]** In the flows report, each flow has the following attributes (columns):

**[0068]** **Errors**. A graphical depiction of the number of errors and warnings as is done in other reports. Includes a link to the error report where the errors belonging to frames in the flow would be shown. Since a flow comprises one or more frames, the errors identified by or associated to any frame in the flow should be included in this summary.

**[0069]** **Sending Node** (name and address). The node that sends the data frames in the flow. This node will also receive TCP acknowledgements from the receiving node.

[0070] **Receiving Node** (name and address). The node that receives the data frames in the flow. This node will also send TCP acknowledgements to the sending node.

[0071] **Data Duration** (seconds): The time period from when the sending node sent the first frame in the flow to the time that the receiving node received the last data frame in the flow. This is related to other fields as follows: Data Duration = (End Data Time – Start Time)

[0072] **Avg Data Rate** (bits/sec). The average data rate during the flow. This is important because flows with low data rate may be worthy of further investigation to determine why the data is not transferred more quickly, particularly if the flow is also longer than most other flows. Computed as (Data Payload Bytes * 8) / (End Data Time – Start Time)

[0073] **Bytes**. The total number of bytes in all of the frames in the flow.

[0074] **Data Payload Bytes**. The sum of the payload bytes for all of the frames in the flow.

[0075] **Frames**. Number of frames in the flow.

[0076] **Data Frames**. Number of data frames in the flow.

[0077] **First Frame**. The sequence number of the first frame in the flow.

[0078] **Last Data Frame**. The sequence number of the last data frame in the flow.

**[0079]** **Last Frame**. The sequence number of the last frame, either data or acknowledgement. If there are TCP acknowledgement frames after the last data frame, then this will differ from the Last Data Frame.

**[0080]** **Start Time**. The time that the first data frame was sent.

**[0081]** **End Data Time**. The time that the last data frame was received.

**[0082]** **End Time**. The time that the last frame (data or acknowledgement) was received. Note that this is normally not important because trailing TCP acknowledgements do not have an impact on the response time.

**[0042]** Other columns that may be included are:

**[0083]** **Data Direction**. Either primary-to-secondary or vice versa. Best indicated with arrows (→ and ←). For flows that are within a capture location, should read "within <capture point>"

**[0084]** **Network Busy Time** (seconds). The total time that one or more frames was in transit during the flow.

RTA Algorithm Details

*Overall Approach*

**[0043]** The RTA functions first at the thread level. Each thread is assumed to be a single-threaded sequence of request/response exchanges between the client of the thread and the server of the thread. It is the responsibility of the

protocol decoder to ensure this requirement. A thread is broken down into periods that fit into the following 5 categories:

**[0085]** **Client Processing Time**.

**[0086]** **Client Sending** (Flow being sent from client to server).

**[0087]** **Server Processing Time**.

**[0088]** **Server Sending** (Flow being sent from server to client)

**[0089]** **Processing interrupted by another thread** (only in the case of multi-tier or overlapping requests at the client)

*Exemplary Embodiment - Single Thread*

**[0044]** RTA concepts can be illustrated in one exemplary embodiment as shown in FIG. 4. FIG. 4 shows a bounce diagram for a typical client/server or Web application. (The application could be anything - a 2-tier SQL application, a web browser to a web server, or even an application using home-grown protocols.)

**[0045]** Assume that all of these frames are in a single thread. In this example the client sends a 2-frame request to the server, the server processes for a bit, and then the server sends a 3-frame reply back to the client. The diagram shows the data frames that would be exchanged as well as TCP acknowledgements that will be seen if the application uses TCP/IP. (TCP is a very dynamic protocol and may not send a TCP acknowledgement for every frame. So the diagram below is only one of the many that could have occurred.)

Node Processing and Sending

**[0046]** The Application Performance Report – Chart would show that the processing time for the client was the sum of the two processing times identified in the figure – the one at the beginning and the one at the end.

**[0047]** The processing time for the server is just the single processing time in the middle of the trace.

**[0048]** The sending time for both nodes is indicated in the figure.

Flows

**[0049]** In this example there are two flows. The first flow is sent from the client to the server and comprises frames 1 through 4. The last data frame in the flow is frame 3. The duration and data duration of the first flow is annotated in FIG. 5. Because frame 4 is a TCP acknowledgement that is sent after the last data frame (3) is sent in the flow, it is not considered a meaningful frame and the network is not considered busy for the time that frame 4 is in transit.

**[0050]** A similar analysis could be made for the second flow in this example. The second flow is sent from the server to the client, and comprises frames 5 through 10. The data duration is from the time frame 5 is sent to the time frame 8 is received.

Network Frame in Transit and Latency Statistics

**[0051]** Again referencing FIG. 4, the times when a frame is in transit can be seen. The TCP acknowledgements that are returned after a flow is

completely received have no impact on the overall response time. Therefore, they are omitted from the network Frame in Transit measure and Latency statistics. In this example frames 4 and 10 do not impact the user-perceived application response time and thus are not included in the network latency measures.

**[0052]** A high network frame in transit time can be an indication that the combination of network latency and the application's sequential request/response behavior is affecting the response time. If you see a high network busy time, you should also look at the network utilization to see if the cause of the high busy time is insufficient bandwidth. This can be easily done in the Performance Report – Chart since the network frame in transit is broken down into components caused by bandwidth and latency. If the bandwidth portion of the bar dominates, then the lack of bandwidth is causing the network to be busy.

*Exemplary Embodiment - Multi-Thread*

Node Processing and Sending Time

**[0053]** Node processing time becomes more complicated when the application is multi-threaded. In the example above, the sum of the node processing times equals the node active times. When there are overlapping node processing times at a node, only one of them is counted and consequently the sum of the individual node processing times can be greater than the overall node processing time.

**[0054]** The user is shown results for both node processing and sending times. Node processing is when the node is definitely processing. Node sending times mean that the node might be processing and consequently this time might contribute to the total response time. Sending times could be caused by other factors such as the receiving node's inability to process incoming data fast enough, insufficient bandwidth in the network, the sending node's inability to make all of the data available quickly enough, a TCP window size that is too small. One can drill down from the node sending time bar to the flows sent by that node. From there, the user should look at the flows and attempt to identify if the time between frames sent by the node was really processing time or not. Often this determination will require some knowledge of the application. In cases where the network is heavily utilized or has high latency, a high node sending time can be caused by the network.

**[0055]** Node Processing Time can tell you how much of the time the node was processing requests (as a server) or processing a reply prior to sending the next request (as a client). The aggregate sending time is important because it can tell you how much of the time the node was trying to send data to other nodes. It is difficult to draw absolute conclusions about what caused the sending time without looking at the individual flows that the node sent.

Node Processing Times

**[0056]** The Node Processing Detail report shows the individual processing times that were detected for each node. The report is sorted initially in

descending duration so that you can easily see the largest individual processing times at the top. Node processing times occur at client nodes prior to the node sending out the first request in a thread, and then within a thread prior to each request that the node sends. Processing times occur at servers from the time the server receives a response until the time that it begins sending a reply.

**[0057]** For further understanding of node processing times, a Node Processing Detail report can be opened on a trace. The window can then be split, and the packet trace placed at the bottom. For each node processing time that is selected, there will be two frames surrounding the processing time. For client processing times, there will be a prior reply (or the beginning of the trace) and the request that the client sends at the end of its processing time. For server processing times, there will be the request (actually the last frame in the request if it is a multi-frame request) and the response (actually the first frame in the response if it is a multi-frame response).



Flows

**[0058]** As discussed previously, there can be many causes for a high node sending time, and the cause of this can be difficult to determine.

Overlapping Threads

**[0059]** Any time two or more threads overlap, more than one node can be processing or sending at a time. Or, a single node could be processing or sending on more than one thread at the same time. These processing and sending times are aggregated by concluding that:

**[0090]** A node is processing if it is processing on one or more threads,

**[0091]** A node is sending if it is sending on one or more threads and is not processing.

*Exemplary Embodiment - Multi-Tier*

**[0060]** FIG. 6 may be used to describe the handling of multi-tier. The multi-tier algorithm works as follows:

**[0092]** At each point in time that a data frame enters a node, the time, frame seq# and its thread is recorded in member variables of the CNode class.

**[0093]** When a client of a node sends out the first frame in a request, there will be a processing time. To determine its type, it is determined if the most recent data frame that arrived at the node was a request frame from another client. If it was, then <MISSING TEXT>.

Conclusion

**[0061]** Thus, an exemplary method and system for performing response time analysis of network performance have been described. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention. Accordingly, the above description and drawings are to be regarded in an illustrative rather than a restrictive sense.